SOFE 3980U
Software Quality

Assignment 1

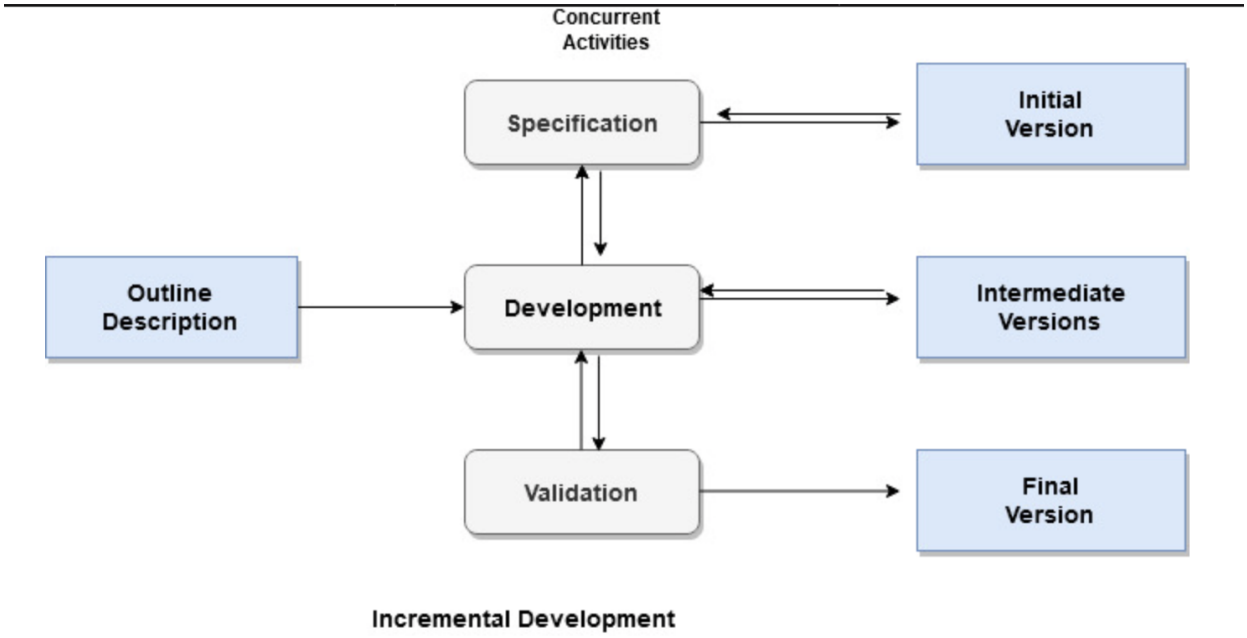
Name: Daniyal Khan (100750029)

Due Date: February, 17th ,2022

Problem A Github Link: (https://github.com/100750029/Software-Quality-Assignment1)

Software engineering process: Incremental development/Agile

The approach of the incremental development/agile process is specification, development and validation are interleaved. You have a choice to follow a plan driven or agile approach. The incremental development process has four main phases: outline description, specification, development, and validation. The last three phases are done through concurrent activities. Here is a diagram which will better illustrate the process.



Incremental Development

https://softschema.com/incremental-model-of-software-development-life-circle/

The reason for choosing the Incremental development/Agile approach is because of some of the benefits it provides:

- Development is taking place in parallel. Different teams can work on different modules.
- Modules can be completed in any order.
- Separation of concerns.Each module is a stand-alone component of the product.
- Deliver working software as soon as possible: with the first module completed (iteration).
- Adaptable to scope shifts. Modules can be added or removed from the product at any time.

- Risks can be identified and managed on a module-by-module basis.
- The expense of adapting to changing customer demands is lower.
- It is much easier to obtain client feedback on previously completed development work.

Outline Description:

Now I'll be going through the different phases in terms of the game I developed in high school. First is the outline description where I'll be going through the game I want to develop, the programming language used, and the requirements of my game. When choosing a game to develop I gained inspiration from a game called Wordle. The programming language I will be using is Python. I ran my code in eclipse with a pydev extension.
Link to wordle game: (https://www.nytimes.com/games/wordle/index.html).

Requirements:

- Randomly generated a code as the word
- Allow the user ten guesses before revealing the code
- Have three different difficulties (easy, normal, hard) with each setting allowing more letters and attempts
- Indicate the right position of a letter and if a letter guessed is in the code
- Show the attempts as the user guesses

Instructions:

Welcome to Code Breaker! In this program, the user will be asked to guess a secret code generated by the computer, given hints after each guess. The user will only be allowed 8, 10, 12 guesses depending on the difficulty in which they must guess the code. There are two types of clues, "b" and "w". The letter "b" means that a color in the user's guess is in the right position and the letter "w" means that a color in the user's guess is in the code but in the wrong position.

Functions:

```
def create_code(characters, length):

    #Creates the list that will store the code
    code = []
    #Adds a random letter from characters to code length amount of times
    for i in range(length):
        code.append(random.choice(characters))

    return code
```

The create_code function returns a list the length of length (int) with any random single characters from characters (str).

Ex.

>>> create_code("Daniyal", 4) = ['D', 'i', 'y', 'l']

>>> create_code("Hello", 3) = ['H', 'H', 'H']

```
def find_fully_correct(answer, guess):

    #Creates a list that will store the "b" clues
    correct_b = []
    #Goes through each index value in answer
    for i in range(len(answer)):
        #Checks if the colour at the index i in guess is
        #the same colour at the index i in answer
        if guess[i] == answer[i]:
            #Adds a "b" to correct_b
            correct_b.append("b")

    return correct_b
```

The find_fully_correct function returns a list containing the letter "b" for each color in the guess (list of strs) that is in the same position as the same color in the answer (list of strs).

Ex.

>>> find_fully_correct(['G', 'B', 'R', 'Y'], ['G', 'R', 'R', 'O']) = ['b', 'b']

>>> find_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O']) = ['b']

>>> find_fully_correct(['P', 'G', 'R', 'O', 'Y'], ['Y', 'G', 'O', 'O', 'Y']) = ['b', 'b', 'b']

```
def remove_fully_correct(list1, list2):

    #Creates the new list
    new_list = []
    #Goes through each index value in list1
    for i in range(len(list1)):
        #Checks if the letter at the index i in list1 is
        #not the same letter at the index i in list 2
        if list1[i] != list2[i]:
            #Adds that letter to new_list
            new_list.append(list1[i])

    return new_list

```

The function remove_fully_correct returns a list that has all the characters in list1 except for those that are in the same position as the same character in list2.

Ex.

>>> remove_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O']) = ['R', 'O', 'P']
>>> remove_fully_correct(['A','B','O','D'], ['D','B','A','D']) = ['A', 'O']
>>> remove_fully_correct(['B', 'B', 'B'], ['A', 'B', 'B']) = ['B']

```
def find_colour_correct(answer, guess):

    #Creates the list that will store all the "w" clues
    correct_w = []
    #Removes all the colours that are in the right position from both answer and guess
    new_guess = remove_fully_correct(guess, answer)
    new_answer = remove_fully_correct(answer, guess)
    #Goes through each colour in new_guess
    for colour in new_guess:
        #Checks if the colour is in new_answer
        if colour in new_answer:
            #Adds a "w" to correct_w
            correct_w.append("w")
            #Removes that colour from new_answer so that it isn't checked twice
            new_answer.remove(colour)

    return correct_w
```

The function find_colour_correct returns a list containing the letter "w" for each character in guess (list of strs) that is also in answer (list of strs) but isn't in the right position.

Ex.

>>> find_colour_correct(['Y','P','G','G'], ['G','P','O','R']) = ['w']
>>> find_colour_correct(['O', 'P', 'P', 'R'], ['O', 'R', 'P', 'P']) = ['w', 'w']
>>> find_colour_correct(['Y','P','G'], ['G','G','O']) = ['w']

```
def display_game(guesses, clues):

    s = 'Guess\tClues\n' + '*' * 20 + '\n'
    for i in range(len(guesses)):
        for j in range(len(guesses[i])):
            s = s + guesses[i][j] + ' '
        s = s + '\t'
        for k in range(len(clues[i])):
            s = s + clues[i][k] + ' '
        s = s + '\n'
    return s
```

The function display_game returns a string to display the current state of the game. The string should contain the headers "Guess" and "Clues" separated by a tab with the next line consisting of 20 "*". Each line after should be of each guess (sub-list) in guesses (list of lists) alongside the corresponding clue (sub-list) from clues (list of lists) which are separated by a tab.

```
def valid(user_guess, valid_characters, guess_size):

    #Checks if the length of user_guess is equal to guess_size
    if len(user_guess) == guess_size:
        #Goes through each character in user_guess
        for character in user_guess:
            #Returns False if the character is not found in valid_characters
            if character not in valid_characters:
                return False
    else:
        return False

    return True
```

The function valid returns True if all the characters in user_guess (list) are in valid_characters (str) and the length of user_guess is equal to guess_size (int).

Ex.

>>> valid(['H', 'E', 'L', 'L', 'O'], "HELOYT", 5) True
>>> valid(['S'], "S", 3) False
>>> valid(['G', 'R', 'R', 'Y'], "GRBYOP", 4) True

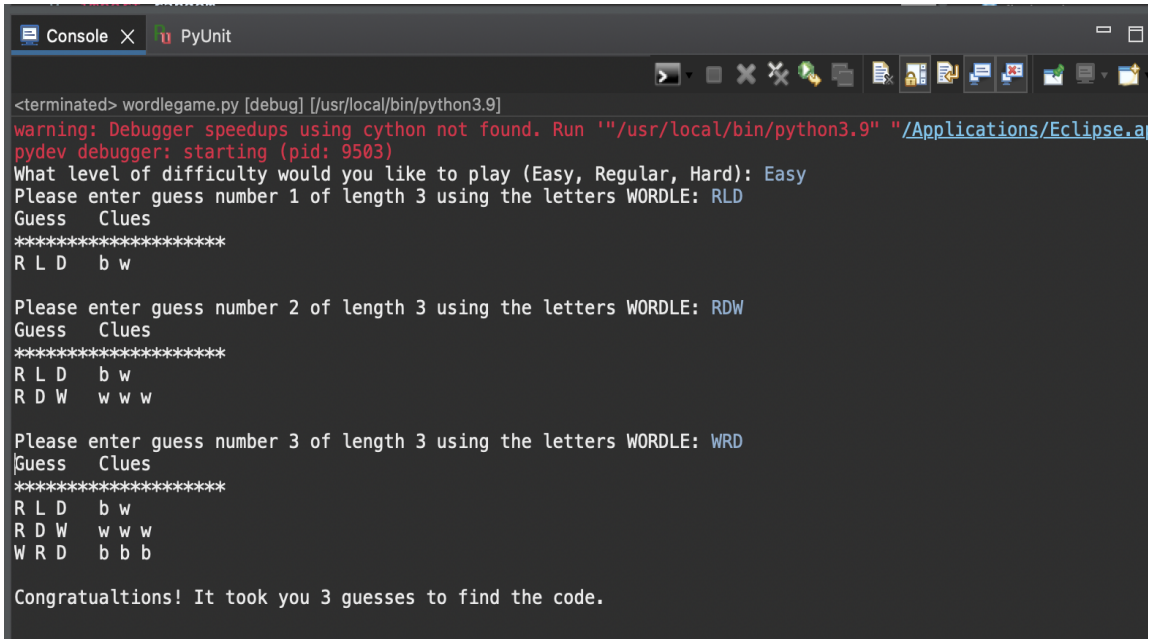
```
def display_clues(guesses, clues, guess):

    #Adds the users guess to a list of their previous guesses
    guesses.append(guess)
    #Stores all the "b" clues for this guess
    num_b = find_fully_correct(secret_code, guess)
    #Stores all the "w" clues for this guess
    num_w = find_colour_correct(secret_code, guess)
    #Puts both clues together into one list
    num_b.extend(num_w)
    #Adds the clues for this guess to a list of the previous clues
    clues.append(num_b)
    print (display_game(guesses, clues))
```

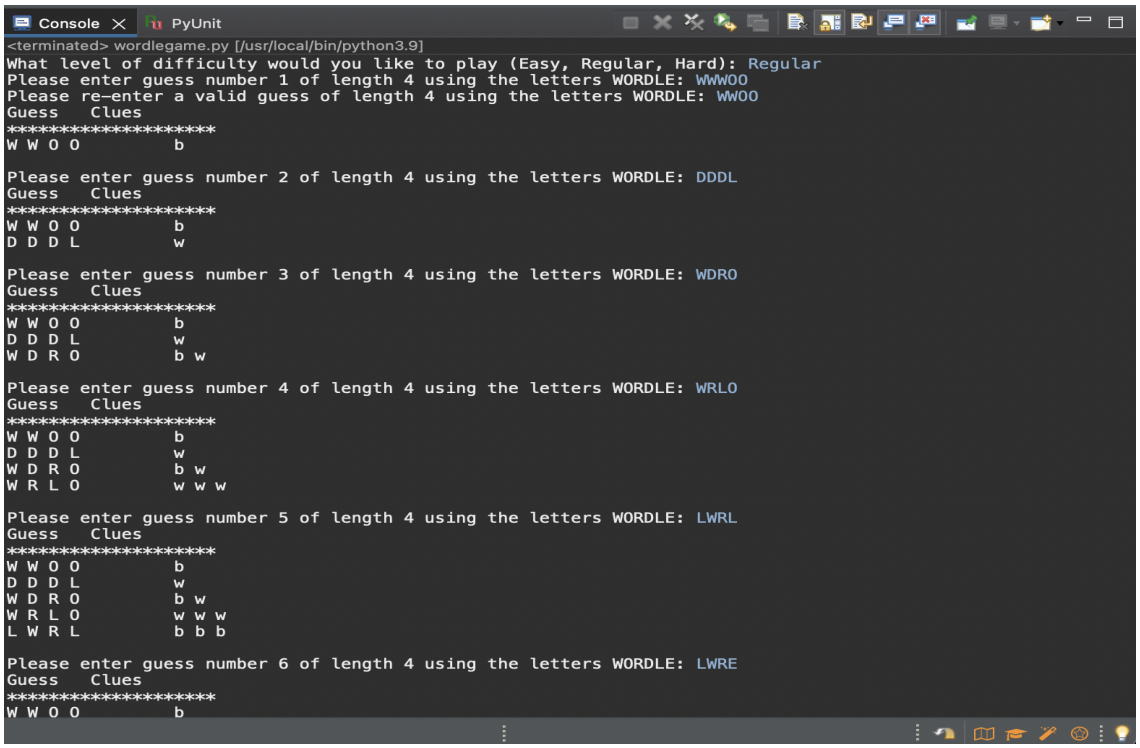
A display_clues function that adds guess (list) to guesses (list of lists) and uses guess to determine the clues that will be added to clues (list of lists) so that they can be used as the arguments for the function display_game(). Prints the function display_game.

Sample Runs:

Easy Mode



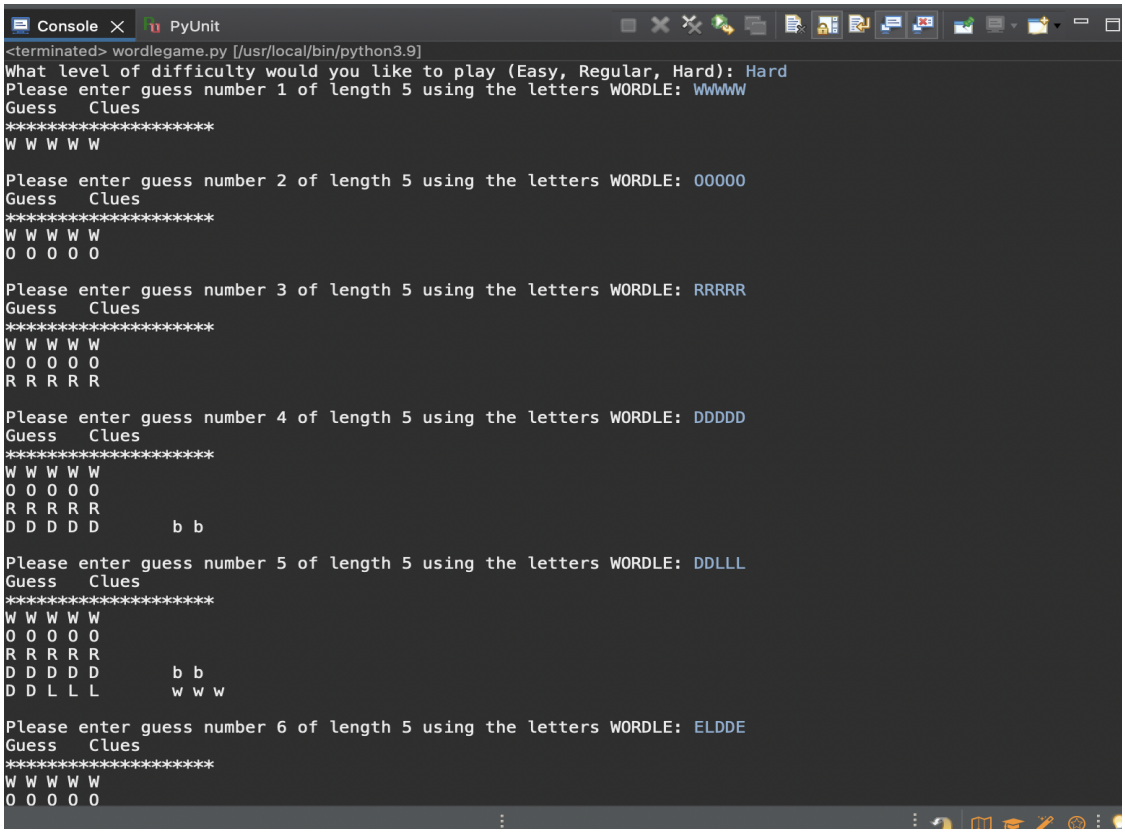
Regular Mode:



```
Please enter guess number 7 of length 4 using the letters WORDLE: LWRR
Guess   Clues
********************
W W O O         b
D D D L         w
W D R O         b w
W R L O         w w w
L W R L         b b b
L W R E         b b b
L W R R         b b b b

Congratualtions! It took you 7 guesses to find the code.
```

Hard Mode:




```
Please enter guess number 7 of length 5 using the letters WORDLE: LEEDD
Guess   Clues
********************
W W W W W
O O O O O
R R R R R
D D D D D       b b
D D L L L       w w w
E L D D E       b w w w w
L E E D D       b b b b b

Congratualtions! It took you 7 guesses to find the code.
```

Problem B

Test Cases:

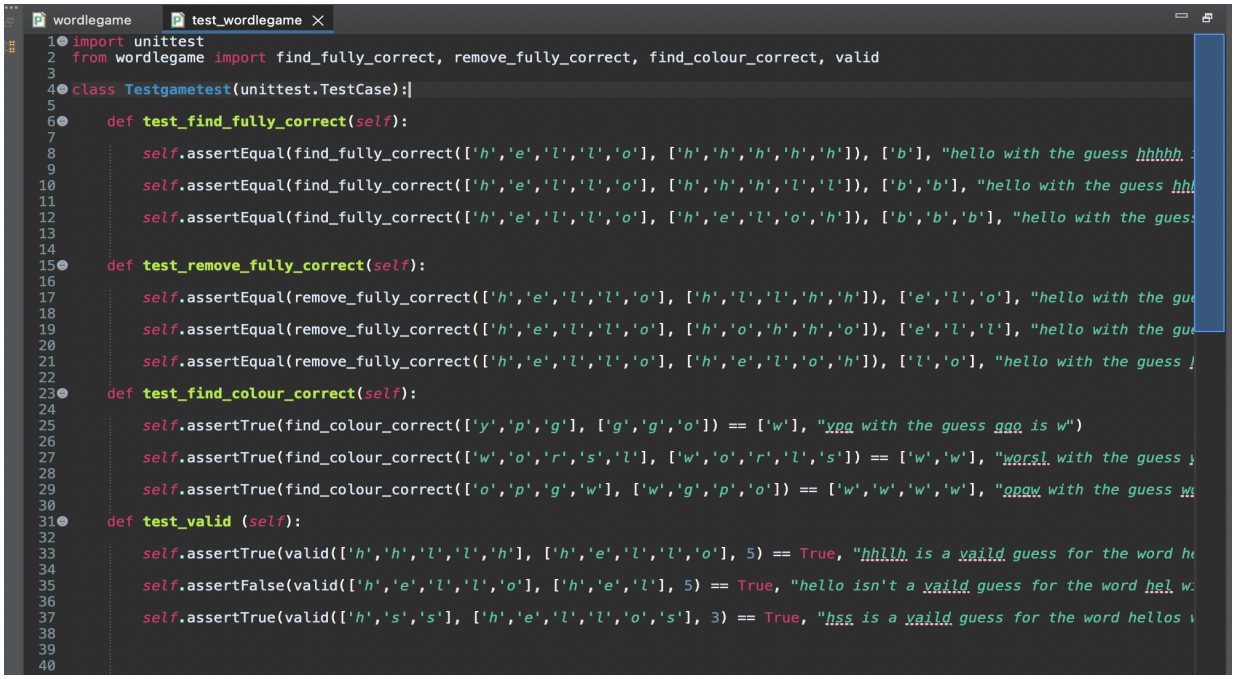
For the test cases I decided to test 4 of my functions. The functions that I will be testing are find_fully_correct, remove_fully_correct, find_colour_correct, and valid. I choose to test these functions because they mostly rely on game logic and I wanted to see if my game runs properly.

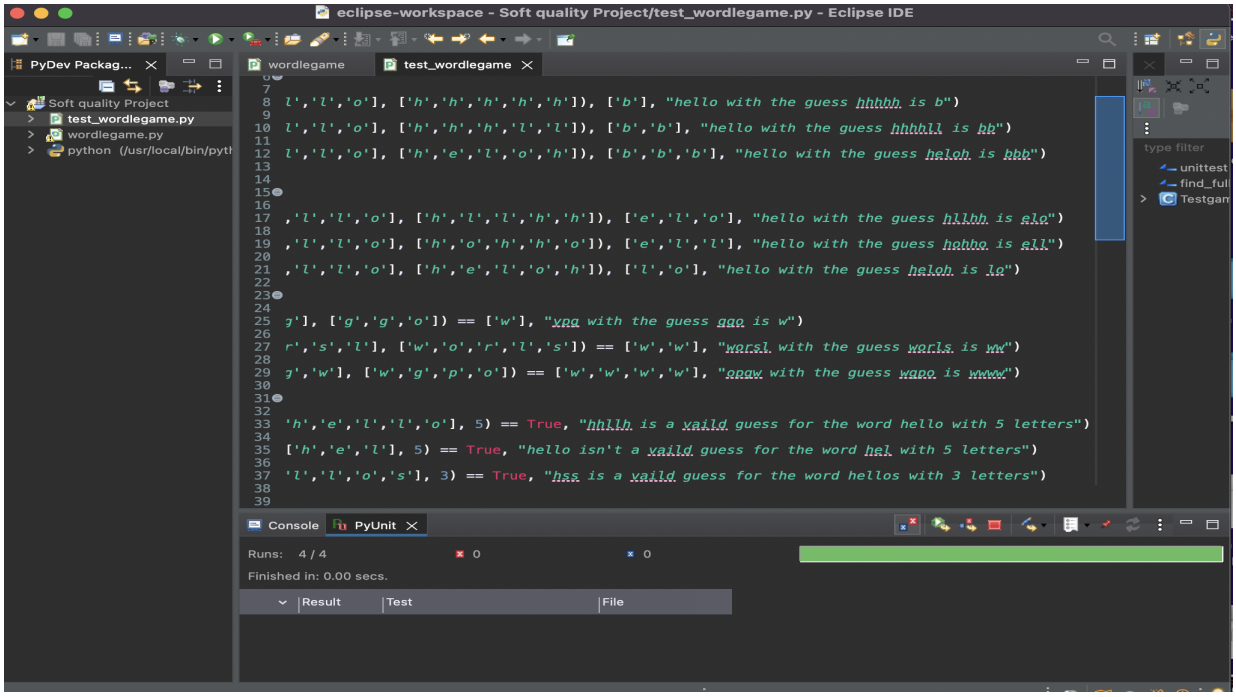
Find_fully_correct: I used assertEquals to test if the guess contains the same letters as the answer and then would print ‘b’.

Remove_fully_correct: I used assertEquals to test if the two lists contain the same letters. It would return the letters that aren't in the same position from list1 compared to list2.

Find_colour_correct: I used assertTrue to test if the guess contained a letter that was in the answer but not in the correct position and then would print ‘w’.

Valid: I used a combination of assertTrue and assertFalse to check if the guess had the appropriate number of letters that the game required. It will return true or false depending on the case.





Challenges Faced:

Now I will go over some of the challenges I faced throughout this assignment:

- The first challenge I faced was deciding which game to make. My first decision was to pick the language I was most comfortable in. So I picked python and then I remembered a game I made in high school, and decided to improve it.

- The next challenge I was faced with was running python on Eclipse. I mostly used to run java on Eclipse, so this was a new experience. I solved this problem by searching online and found out that I needed to install the pydev plugin to write in python.

- Another challenge I faced was writing test cases in python. We just started learning about test cases and the test cases we did were using junit. For python you have to use pyunit which is similar to junit but with different syntax. I learned about pyunit and how to write the test cases from some youtube videos.